Create with Code

Unit 1 Lesson Plans

unity

1.1 Start your 3D Engines

Steps:

Example of project by end of lesson

Length: 70 minutes

Overview: In this lesson, you will create your very first game project in Unity Hub. You will choose and position a vehicle for the player to drive and an obstacle for them to hit or avoid. You will also set up a camera for the player to see through, giving them a perfect view of the scene. Throughout this process, you will learn to navigate the Unity Editor and grow comfortable moving around in 3D Space. Lastly, you will customize your own window layout for the Unity Editor.

Project Outcome: You will have a vehicle and obstacle positioned on the road and the camera set up perfectly behind the vehicle. You will also have a new custom Unity layout, perfectly optimized for editing.

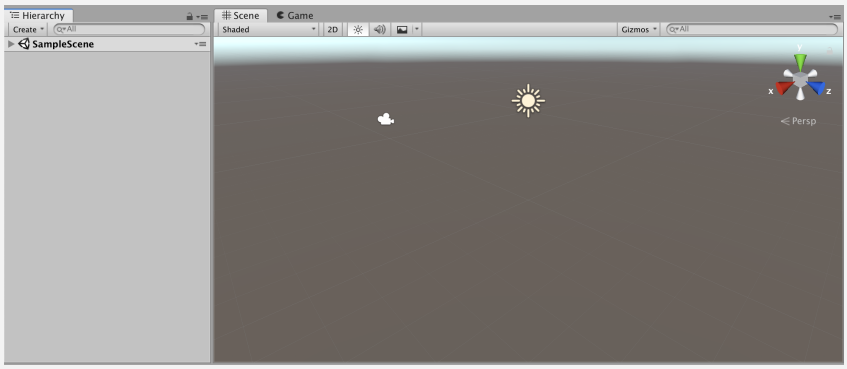
Learning Objectives: By the end of this lesson, you will be able to:
- Create a new project through Unity Hub
- Navigate 3D space and the Unity Editor comfortably
- Add and manipulate objects in the scene to position them where you want
- Position a camera in an ideal spot for your game
- Control the layout of Unity Editor to suit your needs

Step 1: Make a course folder and new project

The first thing we need to do is create a folder that will hold all of our course projects, then create a new Unity project inside it for Prototype 1.

1. On your **desktop** (or somewhere else you will remember), *Right-click > create **New Folder***, then name it "Create with Code"
2. Open **Unity Hub** and select the **Projects** tab from the left sidebar
3. Select **New** to create a new project, using one of the supported versions of Unity (2018.4LTS, 2019.4LTS, or 2020.3LTS)
4. Select the **3D** template, name the project "Prototype 1", and set the location to the new "**Create with Code**" folder.
5. Select **Create**, then wait for Unity to open your new project

- **Don't worry:** Unity might take a while to open, so just give it some time

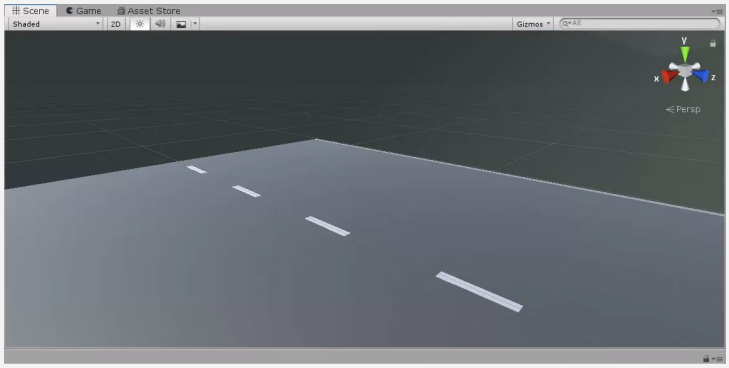


Step 2: Import assets and open Prototype 1

Now that we have an empty project open, we need to import the assets for Prototype 1 and open the scene

1. Click to download the Prototype 1 Starter Files, then **extract** the compressed folder.
   Windows: Right-click on the file > Extract All
   Mac: Double-click on the file
2. From the top menu in Unity, select **Assets** > **Import Package** > **Custom Package**, then navigate to the folder you extracted and select the Prototype-1_Starter-Files.unitypackage file.
3. In the **Import Unity Package** window that pops up, select **Import** and wait for the assets to import.
4. In the **Project** window, in *Assets* > *Scenes* > double-click on the **Prototype 1 scene** to open it
5. Delete the **Sample Scene** without saving
6. **Right-click + drag** to look around at the start of the road

- **Warning:** You're free to look around, but don't try moving yet
- **Warning:** Be careful playing with this interface, don't click on anything else yet
- **New Concept:** Project Window

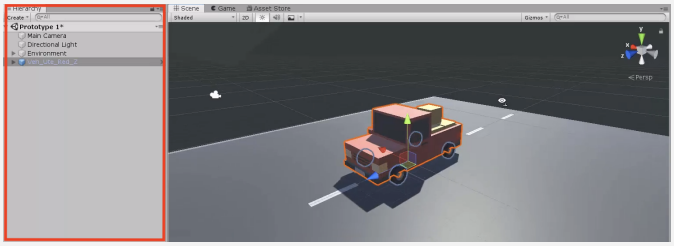


Step 3: Add your vehicle to the scene

Since we're making a driving simulator, we need to add our own vehicle to the scene.

1. In the **Project Window**, open *Assets > Course Library > Vehicles*, then drag a vehicle into the **Hierarchy**
2. **Hold right-click + WASD** to fly to the vehicle, then try to rotate around it
3. With the vehicle selected and your mouse in the Scene view, **Press F** to focus on it
4. then use the **scroll wheel** to zoom in and out and **hold the scroll wheel** to pan
5. **Hold alt+left-click** to rotate around the focal point or **hold alt+right-click** to zoom in and out
6. If anything goes wrong, press **Ctrl/Cmd+Z** to Undo until it's fixed

- **New:** Hierarchy
- **New: Undo** (Cmd/Ctrl + Z) and **Redo** (Cmd+Shift+Z / Ctrl+Y)
- **Warning:** Mouse needs to be in scene view for F/focus to work
- **New Technique:** Scroll Wheel for Zoom and Pan



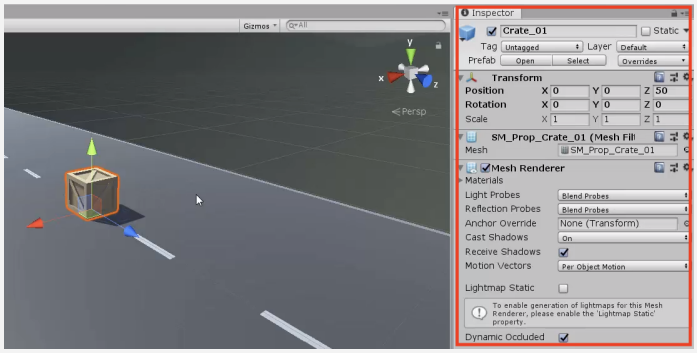
Step 4: Add an obstacle and reposition it

The next thing our game needs is an obstacle! We need to choose one and position it in front of the vehicle.

1. Go to *Course Library* > *Obstacles* and **drag an obstacle** directly into the **Scene view**
2. In the Inspector for your obstacle, in the top-right of the Transform component, click the more options button > ***Reset Position***

   Note: The more options button may appear as three vertical dots or a gear icon, depending on your version of Unity
3. In the **Inspector,** change the **XYZ Location** to **x=0, y=0, z=25**
4. In the Hierarchy, *Right-click* > ***Rename*** your two objects as "Vehicle" and "Obstacle"

- **New Concept:** XYZ location, rotation and scale
- **New Concept:** Inspector

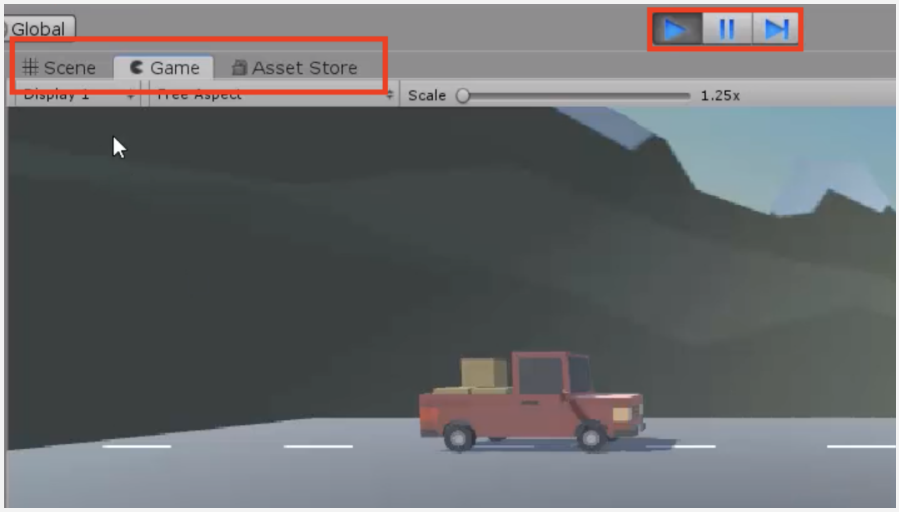


Step 5: Locate your camera and run the game

Now that we've set up our vehicle and obstacle, let's try running the game and looking through the camera.

1. Select the **Camera** in the hierarchy, then **press F** to focus on it
2. Press the **Play button** to run your Game, then press Play again to **stop** it

- **New Concept:** Game View vs Scene View
- **New Technique:** Stop/Play (Cmd/Ctrl + P)

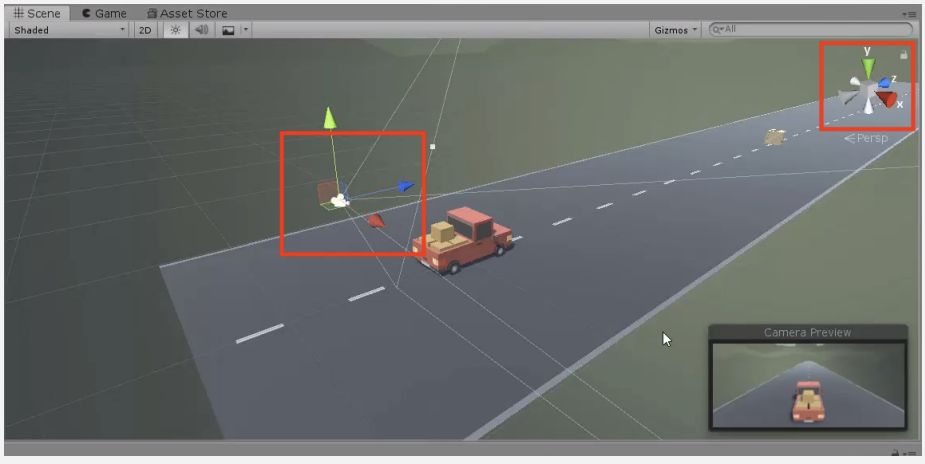


Step 6: Move the camera behind the vehicle

In order for the player to properly view our game, we should position and angle the camera in a good spot behind the vehicle

1. Use the **Move** and **Rotate tools** to move the camera behind the vehicle looking down on it
2. **Hold Ctrl/Cmd** to move the camera by whole units

- **New Technique:** Snapping (Cmd/Ctrl + Drag)
- **New Concept:** Rotation on the XYZ Axes



Step 7: Customize the interface layout

Last but not least, we need to customize the Unity Editor layout so that it's perfect for editing our project.

1. In the top-right corner, change the layout from "Default" to "**Tall**",
2. Move **Game view** beneath Scene view
3. In the **Project** window, click on the little drop-down menu in the top-right and choose "**One-column layout**"
4. In the layout Dropdown, **save a new Layout** and call it "My Layout"

- **New Concept:** Layouts

Lesson Recap

New Functionality	• Project set up with assets imported • Vehicle positioned at the start of the road • Obstacle positioned in front of the vehicle • Camera positioned behind vehicle
New Concepts and Skills	• Create a new project • Import assets • Add objects to the scene • Game vs Scene view • Project, Hierarchy, Inspector windows • Navigate 3D space • Move and Rotate tools • Customize the layout
Next Lesson	• We'll really make this interactive by writing our first line of code in C# to make the vehicle move and have it collide with other objects in the scene

unity

1.2 Pedal to the Metal

Steps:

Step 1: Create and apply your first script

Step 2: Add a comment in the Update() method

Step 3: Give the vehicle a forward motion

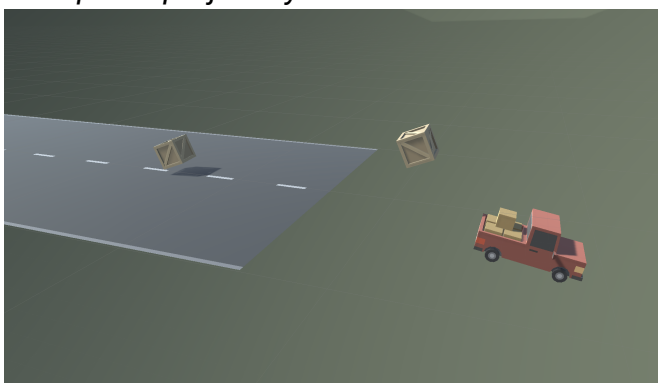
Step 4: Use a Vector3 to move forward

Step 5: Customize the vehicle's speed

Step 6: Add RigidBody components to objects

Step 7: Duplicate and position the obstacles

Example of project by end of lesson



Length: 70 minutes

Overview: In this lesson you will make your driving simulator come alive. First you will write your very first lines of code in C#, changing the vehicle's position and allowing it to move forward. Next you will add physics components to your objects, allowing them to collide with one another. Lastly, you will learn how to duplicate objects in the hierarchy and position them along the road.

Project Outcome: You will have a moving vehicle with its own C# script and a road full of objects, all of which may collide with each other using physics components.

Learning Objectives: By the end of this lesson, you will be able to:

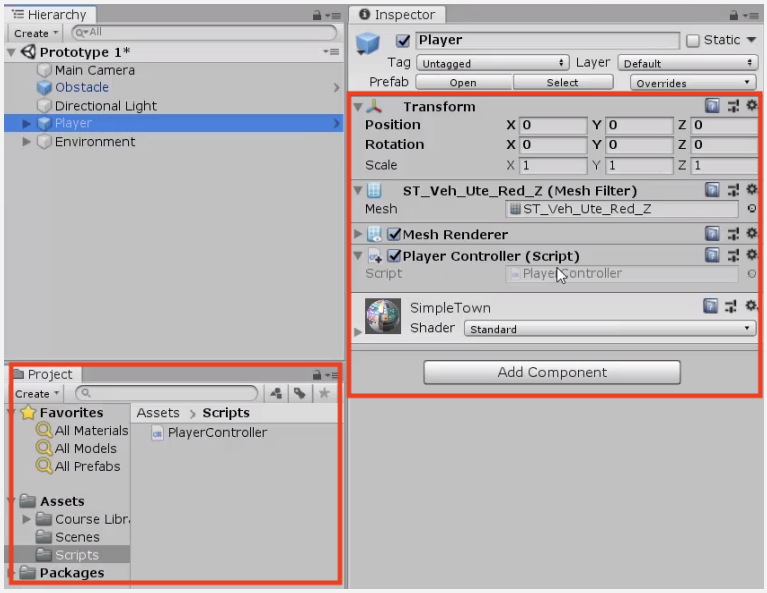
- Create C# scripts and apply them to objects
- Use Visual Studio and a few of its basic features
- Write comments to make your code more readable
- Utilize fundamental C# methods and classes like transform.Translate and Vector3
- Add Rigidbody and Collider components to allow objects to collide realistically
- Duplicate objects in the hierarchy to populate your scene

Step 1: Create and apply your first script

We will start this lesson by creating our very first C# script that will control the vehicle's movement.

1. In the Project window, *Right-click > Create* ***> Folder*** named "Scripts"
2. In the "Scripts" folder, *Right-click > Create >* ***C# Script*** named "PlayerController"
3. **Drag** the new script onto the **Vehicle object**
4. **Click** on the Vehicle object to make sure it was added as a **Component** in the Inspector

- **New Concept:** C# Scripts
- **Warning:** Type the script name as soon as the script is created, since it adds that name to the code. If you want to edit the name, just delete it and make a new script
- **New Concept:** Components



Step 2: Add a comment in the Update() method

In order to make the vehicle move forward, we have to first open our new script and get familiar with the development environment.

1. **Double-click** on the script to open it in **Visual Studio**
2. In the ***Update()*** method, add a comment that you will: ***// Move the vehicle forward***

- **New:** Start vs Update functions
- **New:** Comments

```
void Update()
{
  // Move the vehicle forward
}
```

Step 3: Give the vehicle a forward motion

Now that we have the comment saying what we WILL program - we have to write a line of code that will actually move the vehicle forward.

1. Under your new comment, type ***transform.tr***, then select **Translate** from the autocomplete menu
2. Type **(**, add **0, 0, 1** between the parentheses, and complete the line with a semicolon (**;**)
3. Press **Ctrl/Cmd + S** to save your script, then run your game to test it

- **New Function:** transform.Translate
- **New Concept:** Parameters
- **Warning:** Don't use decimals yet. Only whole numbers!

```
void Update()
{
  // Move the vehicle forward
  transform.Translate(0, 0, 1);
}
```

Step 4: Use a Vector3 to move forward

We've programmed the vehicle to move along the Z axis, but there's actually a cleaner way to code this.

1. **Delete** the 0, 0, 1 you typed and use auto-complete to **replace it** with ***Vector3.forward***

- **New Concept:** Documentation
- **New Concept:** Vector3
- **Warning:** Make sure to save time and use Autocomplete! Start typing and VS Code will display a popup menu with recommended code.

```
void Update()
{
  // Move the vehicle forward
  transform.Translate(0, 0, 1 Vector3.forward);
}
```

Step 5: Customize the vehicle's speed

Right now, the speed of the vehicle is out of control! We need to change the code in order to adjust this.

1. Add **** Time.deltaTime*** and run your game
2. Add **** 20*** and run your game

- **New Concept:** Math symbols in C#
- **New Function:** Time.deltaTime

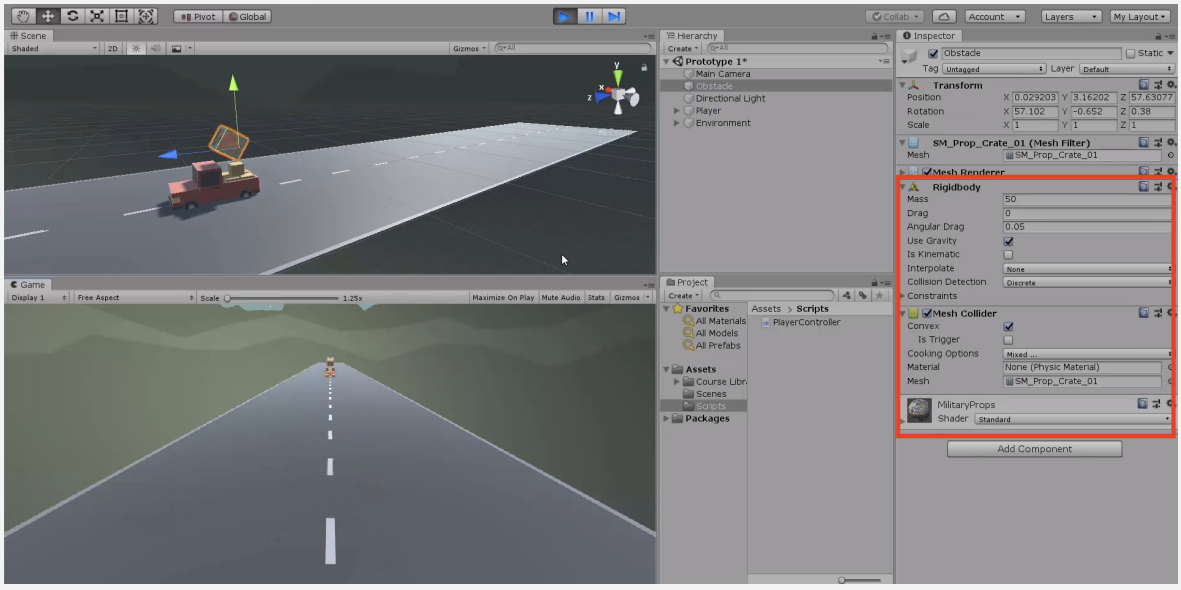
```
void Update()
{
  // Move the vehicle forward
  transform.Translate(Vector3.forward * Time.deltaTime * 20);
}
```

Step 6: Add RigidBody components to objects

Right now, the vehicle goes right through the box! If we want it to be more realistic, we need to add physics.

1. Select the **Vehicle**, then in the hierarchy click **Add Component** and select **RigidBody**
2. Select the **Obstacle**, then in the hierarchy click **Add Component** and select **RigidBody**
3. In the RigidBody component properties, increase the **mass** of vehicle and obstacle to be about what they would be in **kilograms** and test again

- **New Concept:** Rigidbody Component
- **New Concept:** Collider Component
- **Tip:** Adjust the mass of the vehicle and the obstacle, and test the collision results

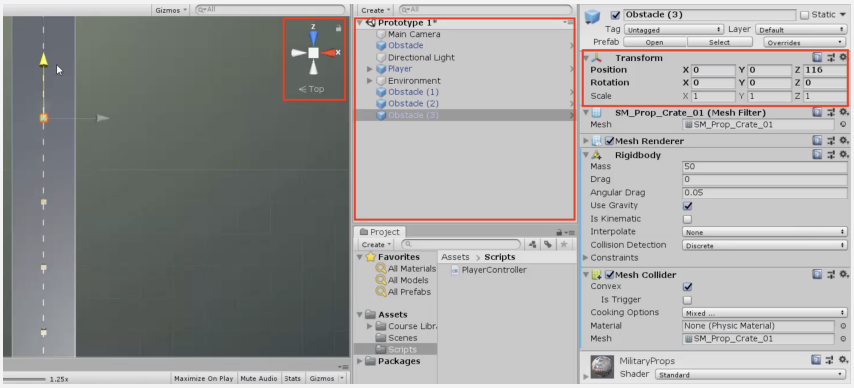


Step 7: Duplicate and position the obstacles

Last but not least, we should duplicate the obstacle and make the road more treacherous for the vehicle.

1. Click and drag your obstacle to the **bottom of the list** in the hierarchy
2. Press **Ctrl/Cmd+D** to duplicate the obstacle and move it down the **Z axis**
3. Repeat this a few more times to create more obstacles
4. After making a few duplicates, select one in the hierarchy and **hold ctrl + click** to select multiple obstacles, then **duplicate** those

- **New Technique:** Duplicate (Ctrl/Cmd+D)
- **Tip:** Try using top-down view to make this easier
- **Tip:** Try using the inspector to space your obstacles exactly 25 apart



Lesson Recap

New Functionality	• Vehicle moves down the road at a constant speed • When the vehicle collides with obstacles, they fly into the air
New Concepts and Skills	• C# Scripts • Start vs Update • Comments • Methods • Pass parameters • Time.deltaTime • Multiply (*) operator • Components • Collider and RigidBody
Next Lesson	• We'll add some code to our camera, so that it follows the player as they drive along the road.

unity

1.3 High Speed Chase

Steps:

Example of project by end of lesson

Length: 50 minutes

Overview: Keep your eyes on the road! In this lesson you will code a new C# script for your camera, which will allow it to follow the vehicle down the road and give the player a proper view of the scene. In order to do this, you'll have to use a very important concept in programming: variables.

Project Outcome: The camera will follow the vehicle down the road through the scene, allowing the player to see where it's going.

Learning Objectives: By the end of this lesson, you will be able to:

- Declare variables properly and understand that variables can be different data types (float, Vector3, GameObject)
- Initialize/assign variables through code or through the inspector to set them with appropriate values
- Use appropriate access modifiers (public/private) for your variables in order to make them easier to change in the inspector
- Use the Update and LateUpdate appropriately in order to call one action after another has already happened

Step 1: Add a speed variable for your vehicle

We need an easier way to change the vehicle's speed and allow it to be accessed from the inspector. In order to do so what we need is something called a variable.

1. In PlayerController.cs, add ***public float speed = 5.0f;*** at the top of the **class**
2. Replace the **speed value** in the Translate method with the **speed variable**, then test
3. **Save** the script, then edit the speed value in the **inspector** to get the speed you want

- **New Concept:** Floats and Integers
- **New Concept:** Assigning Variables
- **New Concept:** Access Modifiers

```
public float speed = 20;

void Update()
{
  transform.Translate(Vector3.forward * Time.deltaTime * 20 speed);
}
```

Step 2: Create a new script for the camera

The camera is currently stuck in one position. If we want it to follow the player, we have to make a new script for the camera.

1. Create a new **C# script** called FollowPlayer and attach it to the **camera**
2. Add ***public GameObject player;*** to the top of the script
3. Select the **Main Camera, then**, **drag** the player object onto the **empty player variable** in the Inspector
4. In ***Update()***, assign the camera's position to the player's position, then test

- **Warning:** Remember to capitalize your script name correctly and rename it as soon as the script is created!
- **Warning:** It's really easy to forget to assign the player variable in the inspector
- **Don't worry:** The camera will be under the car... weird! We will fix that soon

```
public GameObject player;

void Update()
{
  transform.position = player.transform.position;
}
```

Step 3: Add an offset to the camera position

We need to move the camera's position above the vehicle so that the player can have a decent view of the game.

1. In the line in the Update method add ***+ new Vector3(0, 5, -7)***, then test

- **New Concept:** Vector3 in place of coordinates
- **Tip:** You need "new Vector3()" because 3 numbers in a row could mean anything
- **New Concept:** FixedUpdate
- **Warning:** Remember to update your comments and maintain their accuracy!

```
public GameObject player;

void Update()
{
  transform.position = player.transform.position + new Vector3(0, 5, -7);
}
```

Step 4: Make the offset into a Vector3 variable

We've fixed the camera's position, but we may want to change it later! We need an easier way to access the offset.

1. At the top of **FollowPlayer.cs**, declare ***private Vector3 offset;***
2. Copy the ***new Vector3()*** code and **assign** it to that variable
3. **Replace** the original code with the ***offset*** variable
4. **Test** and **save**

- **Don't worry:** Pay no mind to the read only warning
- **Tip:** Whenever possible, make variables! You never want hard values in the middle of your code

```
public GameObject player;
private Vector3 offset = new Vector3(0, 5, -7);

void Update()
{
  transform.position = player.transform.position + new Vector3(0, 5, -7) offset;
}
```

Step 5: Smooth the Camera with LateUpdate

You may have noticed that the camera is kind of jittery as the car drives down the road - let's fix that.

1. Test your prototype to notice the jittering camera as the vehicle drives.
2. In **FollowPlayer.cs**, replace ***Update()*** with ***LateUpdate()*.**
3. **Save** and **test** to see if the camera is less jittery.

- **New Concept:** LateUpdate is called after the Update method, which allows to more smoothly follow the player.

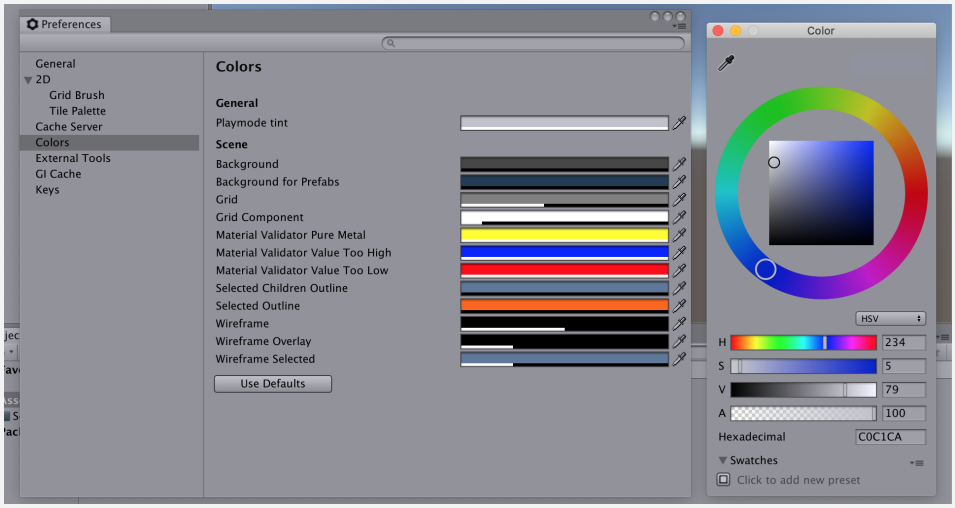
```
void LateUpdate()
{
  transform.position = player.transform.position + offset;
}
```

Step 6: Edit the playmode tint color

If we're going to be creating and editing variables, we need to make sure we don't accidentally try to make changes when in "Play mode"

1. From the top menu, *go to Edit > Preferences* (**Windows**) or *Unity > Preferences* (**Mac**)
2. In the left menu, choose **Colors**, then edit the "**Playmode tint**" color to have a *slight* color
3. **Play** your project to test it, then close your preferences

- **Tip:** Try editing a variable in play mode, then stopping - it will revert
- **Warning:** Don't go crazy with the colors or it will be distracting



Lesson Recap

New Functionality	• Camera follows the vehicle down the road at a set offset distance
New Concepts and Skills	• Variables • Data types • Access Modifiers • Declare and initialize variables • LateUpdate
Next Lesson	• In the next lesson, we'll add our last lines of code to take control of our car and be able to drive it around the scene.

unity

1.4 Step into the Driver's Seat

Steps:

Step 1: Allow the vehicle to move left/right

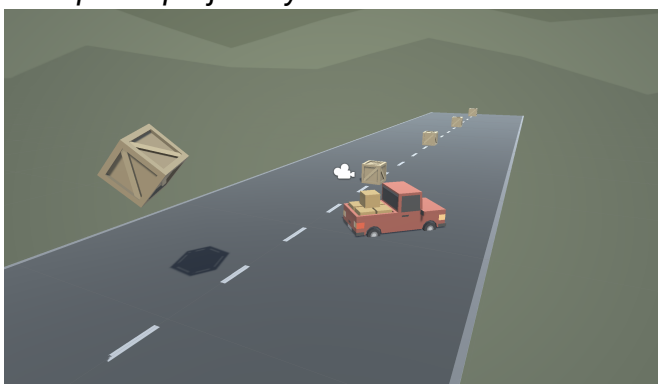
Step 2: Base left/right movement on input

Step 3: Take control of the vehicle speed

Step 4: Make vehicle rotate instead of slide

Step 5: Clean your code and hierarchy

Example of project by end of lesson



Length: 50 minutes

Overview: In this lesson, we need to hit the road and gain control of the vehicle. In order to do so, we need to detect when the player is pressing the arrow keys, then accelerate and turn the vehicle based on that input. Using new methods, Vectors, and variables, you will allow the vehicle to move forwards or backwards and turn left to right.

Project Outcome: When the player presses the up/down arrows, the vehicle will move forward and backward. When the player presses the left/right arrows, the vehicle will turn.

Learning Objectives: By the end of this lesson, you will be able to:

- Gain user input with Input.GetAxis, allowing the player to move in different ways
- Use the Rotate function to rotate an object around an axis
- Clean and organize your hierarchy with Empty objects

Step 1: Allow the vehicle to move left/right

Until now, the vehicle has only been able to move straight forward along the road. We need it to be able to move left and right to avoid the obstacles.

1. At the top of PlayerController.cs, add a ***public float turnSpeed;*** variable
2. In ***Update()***, add ***transform.Translate(Vector3.right * Time.deltaTime * turnSpeed);***
3. Run your game and use the ***turnSpeed*** **variable slider** to move the vehicle left and right

- **New Function:** Vector3.right

```
public float turnSpeed;

void Update()
{
  transform.Translate(Vector3.forward * Time.deltaTime * speed);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed);
}
```

Step 2: Base left/right movement on input

Currently, we can only control the vehicle's left and right movement in the inspector. We need to grant some power to the player and allow them to control that movement for themselves.

1. From the top menu, click **Edit** > **Project Settings**, select **Input Manager** in the left sidebar, then expand the **Axes** fold-out to explore the inputs.
2. In **PlayerController.cs**, add a new ***public float horizontalInput*** variable
3. In ***Update***, assign ***horizontalInput = Input.GetAxis("Horizontal");***, then test to see it in inspector
4. Add the ***horizontalInput*** variable to your left/right **Translate method** to gain control of the vehicle
5. In the Inspector, edit the ***turnSpeed*** and ***speed*** variables to tweak the feel

- **New:** Input.GetAxis
- **Tip:** Edit > Project Settings > Input and expand the Horizontal Axis to show everything about it
- **Warning:** Spelling is important in string parameters. Make sure you spell and capitalize "Horizontal" correctly!

```
public float horizontalInput;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");

  transform.Translate(Vector3.forward * Time.deltaTime * speed);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```

Step 3: Take control of the vehicle speed

We've allowed the player to control the steering wheel, but we also want them to control the gas pedal and brake.

1. Declare a new public ***forwardInput*** variable
2. In ***Update***, assign ***forwardInput = Input.GetAxis("Vertical");***
3. Add the ***forwardInput*** variable to the **forward Translate method**, then test

- **Tip:** It can go backwards, too!
- **Warning:** This is slightly confusing with forwardInput and vertical axis

```
public float horizontalInput;
public float forwardInput;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
  forwardInput = Input.GetAxis("Vertical");

  transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```

Step 4: Make vehicle rotate instead of slide

There's something weird about the vehicle's movement… it's slides left to right instead of turning. Let's allow the vehicle to turn like a real car!

1. In ***Update***, call ***transform.Rotate(Vector3.up, horizontalInput)***, then test
2. **Delete** the line of code that **translates Right**, then test
3. Add **** turnSpeed * Time.deltaTime***, then test

- **New:** transform.Rotate
- **Tip:** You can always trust the official Unity scripting API documentation

```
void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
  forwardInput = Input.GetAxis("Vertical");

  transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
  transform.Rotate(Vector3.up, turnSpeed * horizontalInput * Time.deltaTime);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```

Step 5: Clean your code and hierarchy

We added lots of new stuff in this lesson. Before moving on and to be more professional, we need to clean our scripts and hierarchy to make them more organized.

1. In the hierarchy, *Right-click > Create Empty* and rename it "Obstacles", then **drag** all the obstacles into it
2. **Initialize** variables with values in **PlayerController**, then make all variables **private** (except for the **player** variables)
3. Use **//** to add **comments** to each section of code

- **New:** Empty Object
- **Tip:** You don't actually need to type "private", it defaults to that
- **Tip:** Comments are important, especially for your future self

```
public private float speed = 20.0f;
public private float turnSpeed = 45.0f;
public private float horizontalInput;
public private float forwardInput;

void Update() {
  horizontalInput = Input.GetAxis("Horizontal");
  forwardInput = Input.GetAxis("Vertical");
  // Moves the car forward based on vertical input
  transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
  // Rotates the car based on horizontal input
  transform.Rotate(Vector3.up, turnSpeed * horizontalInput * Time.deltaTime);
}
```

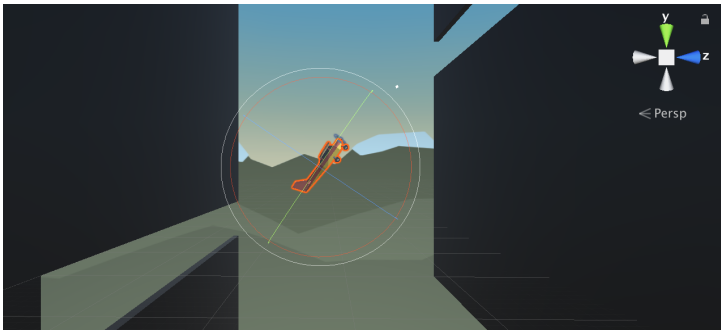
Lesson Recap

New Functionality	• When the player presses the up/down arrows, the vehicle will move forward and backward • When the player presses the left/right arrows, the vehicle turns
New Concepts and Skills	• Empty objects • Get user input • Translate vs Rotate

unity

Challenge 1

Plane Programming



Challenge Overview: Use the skills you learned in the driving simulation to fly a plane around obstacles in the sky. You will have to get the user's input from the up and down arrows in order to control the plane's pitch up and down. You will also have to make the camera follow alongside the plane so you can keep it in view.

Challenge Outcome:
- The plane moves forward at a constant rate
- The up/down arrows tilt the nose of the plane up and down
- The camera follows along beside the plane as it flies

Challenge Objectives: In this challenge, you will reinforce the following skills/concepts:
- Using the Vector3 class to move and rotate objects along/around an axis
- Using Time.deltaTime in the Update() method to move objects properly
- Moving and rotating objects in scene view to position them the way you want
- Assigning variables in the inspector and initializing them in code
- Implementing Input variables to control the movement/rotation of objects based on User input

Challenge Instructions:
- Open your **Prototype 1** project
- **Download** the "Challenge 1 Starter Files" from the Tutorial Materials section, then double-click on it to **Import**
- In the *Project Window > Assets > Challenge 1 >* ***Instructions*** folder, use the Outcome video as a guide to complete the challenge

Challenge		Task	Hint
1	The plane is going backwards	Make the plane go forward	`Vector3.back` makes an object move backwards, `Vector3.forward` makes it go forwards
2	The plane is going too fast	Slow the plane down to a manageable speed	If you multiply a value by `Time.deltaTime`, it will change it from 1x/frame to 1x/second
3	The plane is tilting automatically	Make the plane tilt only if the user presses the up/down arrows	In PlayerControllerX.cs, in Update(), the `verticalInput` value is assigned, but it's never actually used in the `Rotate()` call
4	The camera is *in front of* the plane	Reposition it so it's beside the plane	For the camera's position, try `X=30, Y=0, Z=10` and for the camera's rotation, try `X=0, Y=-90, Z=0`
5	The camera is not following the plane	Make the camera follow the plane	In FollowPlayerX.cs, neither the plane nor offset variables are assigned a value - assign the `plane` variable in the camera's inspector and assign the `offset = new Vector3(30, 0, 10)` in the code

Bonus Challenge		Task	Hint
X	The plane's propeller does not spin	Create a script that spins the plane's propeller	There is a "Propeller" child object of the plane - you should create a new "SpinPropellerX.cs" script and make it rotate every frame around the Z axis.

Challenge Solution

1 In PlayerControllerX.cs, in Update, change *Vector3.back* to *Vector3.forward*

```
// move the plane forward at a constant rate
 transform.Translate(Vector3.back.forward * speed);
```

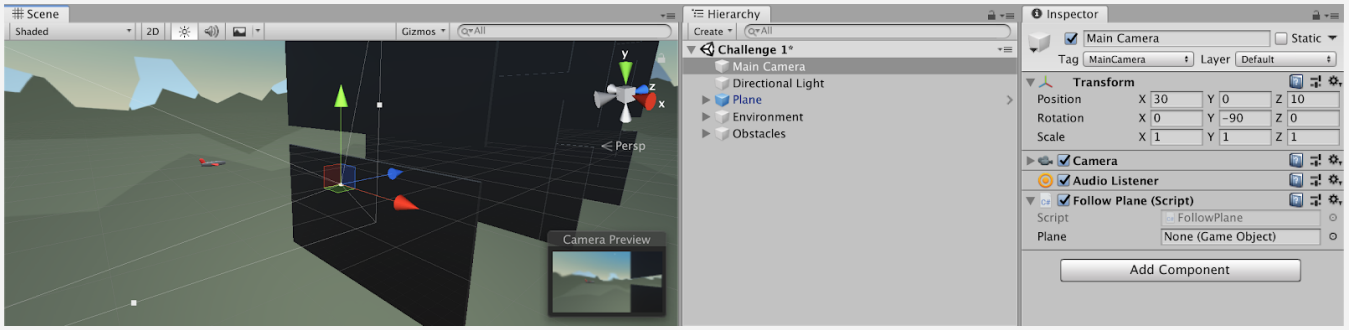
2 In PlayerControllerX.cs, in Update, add * *Time.deltaTime* to the Translate call

```
// move the plane forward at a constant rate
 transform.Translate(Vector3.forward * speed * Time.deltaTime);
```

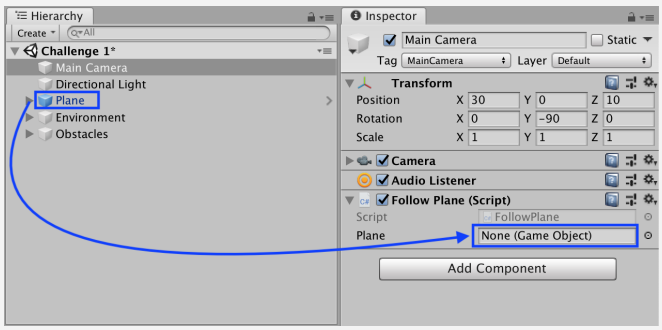
3 In PlayerControllerX.cs, include the *verticalInput* variable to the Rotate method:

```
// tilt the plane up/down based on up/down arrow keys
transform.Rotate(Vector3.right * rotationSpeed * verticalInput * Time.deltaTime);
```

4 Change the camera's position to (30, 0, 10) and its rotation, to (0, -90, 0)



5 To assign the *plane* variable, select **Main Camera** in the hierarchy, then drag the **Plane** object onto the "Plane" variable in the inspector

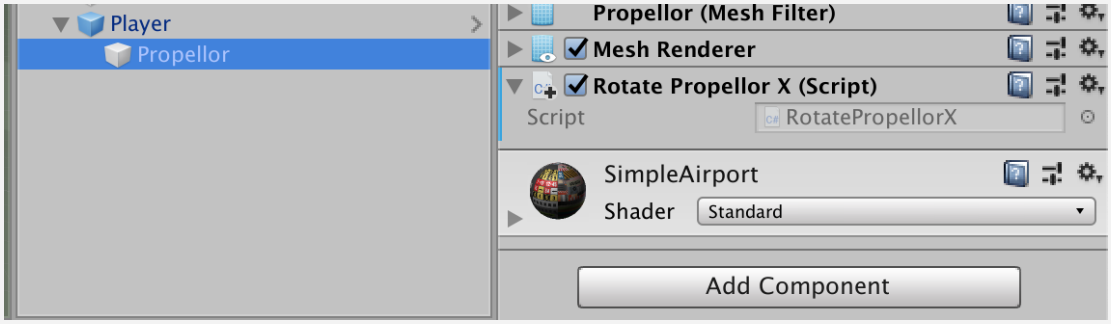


To assign the *offset variable,* add the value as a new Vector3 at the top of FollowPlane.cs:

```
private Vector3 offset = new Vector3(30, 0, 10);
```

Bonus Challenge Solution

X1 Create a new Script called “SpinPropellerX.cs” and attach it to the “Propellor” object (which is a child object of the Plane):



X2 In RotatePropellerX.cs, add a new propellorSpeed variable and Rotate the propeller on the Z axis

```
private float propellorSpeed = 1000;

void Update() {
  transform.Rotate(Vector3.forward, propellorSpeed * Time.deltaTime);
}
```

unity

Unit 1 Lab

Project Design Document

Steps:

Step 1: Understand what a Personal Project is

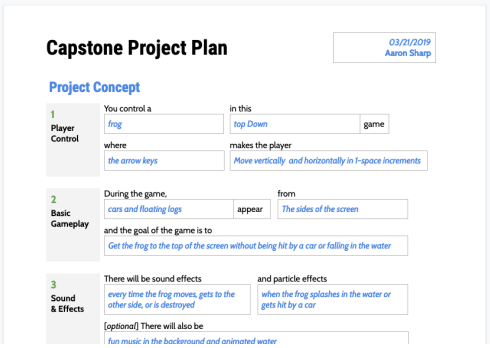
Step 2: Review Design Doc examples

Step 3: Complete your Project Concept V1

Step 4: Complete your Project Timeline

Step 5: Complete your MVP sketch

Example of progress by end of lab



Length: 60 minutes

Overview: In this first ever Lab session, you will begin the preliminary work required to successfully create a personal project in this course. First, you'll learn what a personal project is, what the goals for it are, and what the potential limitations are. Then you will take the time to come up with an idea and outline it in detail in your Design Document, including a timeline for when you hope to complete certain features. Finally, you will take some time to draw a sketch of your project to help you visualize it and share your idea with others.

Project Outcome: The Design Document will be filled out, including the concept, the timeline, and a preliminary sketch of the minimum viable product.

Learning Objectives: By the end of this lab, you will be able to:

- Come up with an idea for a project with a scope appropriate to your time and available resources
- Think through a project's concept in order to better understand its requirements
- Plan out a project's milestones with due dates to better understand the production cycle and to hold yourself more accountable
- Create a simple sketch / storyboard in order to better communicate your ideas

Step 1: Understand what a Personal Project is

Before we get started on our personal projects, we should make sure we understand our primary goals.

Explain

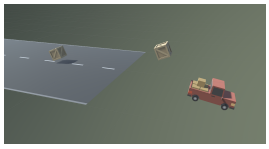
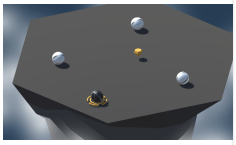
What **Personal Projects** (PP's) are:

- Projects they will be working on on their own with less direct instruction
- A chance to create a project they really care about with their own creative choices
- An opportunity to apply and solidify skills they learned in lessons and challenges

Demo

The **Core Functionality** and skills they will learn from each of the 5 Units by showcasing completed versions of each Prototype:

1. Driving Simulation: **player control** through user input
2. Feed the Animals: **basic gameplay** by spawning random objects on an interval and trying to collect them, avoid them, or fire projectiles at them
3. Run and Jump: **sound and effects,** and animation (of background or player)
4. Sumo Battle: **gameplay mechanics**, powerups and/or increasing difficulty
5. Quick Click: **user interface** with title screen, game over screen, and score display

Unit 1	Unit 2	Unit 3	Unit 4	Unit 5
				

Explain

Goal / Evaluation of the PP's are based on:

- Completeness - how much of what you set out to complete did you actually finish
- Uniqueness / Application - how much did you add new design and dev features, extending and applying your skills in novel and creative ways

NOTE - These two priorities are at odds and it's up to you to find the balance

Explain

You just need a **Minimum Viable Product (an MVP)** - doesn't have to be polished

- Definition: a product with just enough features to satisfy early customers, and to provide feedback for future product development
- This will allow them to focus on the core of the project and not get distracted by flashy features and graphics that don't matter as much

Warning

There will be a **temptation to try and do too much** that is completely different from what anything in the course (e.g. "I want to make "Madden + Facebook + Google!")

- There's *lots* of time to try and do really ambitious crazy projects in the future, but for now on this first project, try to stick closely to the core functionality you're learning
- The *only* limitation is time - with enough time, they could make anything!

Discuss

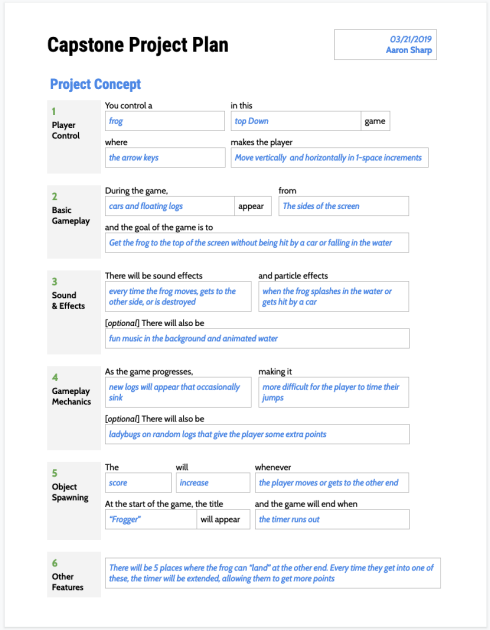
Make sure students understand what the Personal Project is, allowing them to ask questions

Step 2: Review Design Doc examples

Now that we have some idea of what a Personal Project is, let's look a couple examples

1. Click on the link to open a new Project Design Doc as either a Google Doc Copy, a Word Document or PDF
2. Think through how you would fill out a design doc for other games

- **Warning**: you will need to be signed into a Google account to be able to make a copy of the Google Doc version
- **Tip:** Search YouTube for "gameplay" of the classic game you want
- **Explanation:** Notice that sections correspond to what you'll be learning with each unit/prototype



Capstone Project Plan

03/21/2019
Aaron Sharp

Project Concept

1 Player Control
You control a: *frog* in this: *top Down* game
where: *the arrow keys* makes the player: *Move vertically and horizontally in 1-space increments*

2 Basic Gameplay
During the game, *cars and floating logs* appear from *The sides of the screen*
and the goal of the game is to: *Get the frog to the top of the screen without being hit by a car or falling in the water*

3 Sound & Effects
There will be sound effects: *every time the frog moves, gets to the other side, or is destroyed* and particle effects: *when the frog splashes in the water or gets hit by a car*
[optional] There will also be: *fun music in the background and animated water*

4 Gameplay Mechanics
As the game progresses, *new logs will appear that occasionally sink* making it *more difficult for the player to time their jumps*
[optional] There will also be: *ladybugs on random logs that give the player some extra points*

5 Object Spawning
The *score* will *increase* whenever *the player moves or gets to the other end*
At the start of the game, the title *"Frogger"* will appear and the game will end when *the timer runs out*

6 Other Features
There will be 5 places where the frog can "land" at the other end. Every time they get into one of these, the timer will be extended, allowing them to get more points

Step 3: Complete your Project Concept V1

Now that we've seen some examples, let's try to come up with our own project concept.

1. Add your **name** and **date** in the top-right corner
2. **Fill in the blanks** for your project concept
3. **Share** your project concept with someone else to make sure it makes sense to them

- **Explanation:** In the Course Library, you've got human characters, animals, vehicles, foods, sports balls, other random things, but you can always use "primitives" as placeholders in a MVP, then go to the Unity Asset store to get real graphics
- **Tip:** This is good opportunity to catch yourself if you're being too ambitious
- **Don't worry:** This is just a best guess right now, if you want to change your project completely next lab, you could

Step 4: Complete your Project Timeline

Now that we know the basic concept of our project, let's figure out how we're going to get it done.

1. Fill in **milestone descriptions** based on your schedule for the course, including self-imposed due dates
2. Add features that will *not* be included in your MVP to the "**Backlog**"

- **Warning:** This is a MVP, so don't be afraid to put objects on backlog that you'll get to in version 2
- **Explanation:** In Lab 2 you will be setting up your project, in Lab 3 you will do basic player movement, in Lab 4 you will add basic gameplay, and Lab 5 you will add graphics - that would be a good start in filling this out
- **Tip:** This will depend heavily on the schedule you're following for this course - you should leave a significant amount of time to work on it at the end when you've completed all 5 units
- **Don't worry:** It will be hard to do this accurately, since you don't know how long things take - this can change
- **Don't worry:** You don't need to use all milestones - can add more or leave blank rows you are not using
- **Tip:** These should be worded as "Completed functionality" - as in: "Frog can move side-to-side based on left/right arrow keys"

Project Timeline

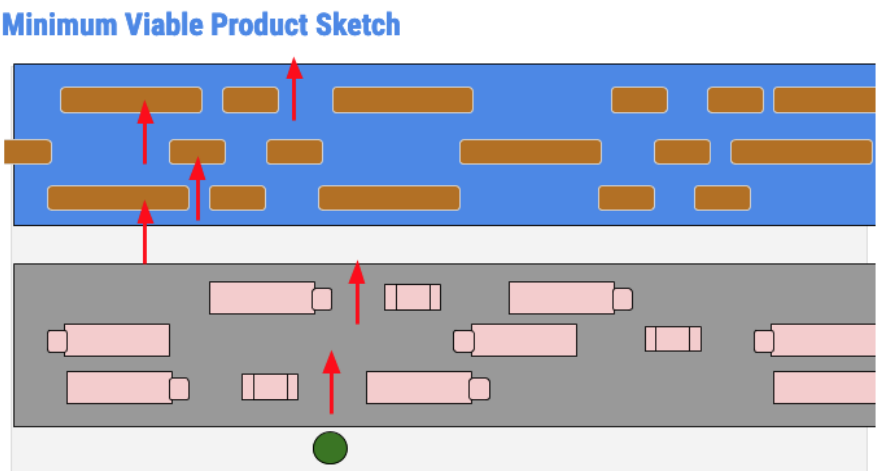
Milestone	Description	Due
#1	- Basic frog movement left to right and side to side	03/01
#2	- Random cars spawning from left and right sides, in 3 discrete lanes - When the player is hit by one of the cars, it will be repositioned at start	04/01
#3	- Logs generating from left/right sides of the screens - The player can ride on these logs and be moved along with them - When player falls in the water (not on log), the are repositioned at start	05/01
Backlog	- 5 discrete areas for the frog to land, each one extending the timer - Random things that pop up and block the 5 landing spots - Ladybug powerup that gives the player extra points and invincibility	09/01

Step 5: Complete your MVP sketch

To help visualize our minimum viable product, it's always helpful to have a sketch.

1. Look at sketch in the **example**
2. Using Google Docs, some other online simple drawing program, or pencil and paper, draw a sketch of your MVP and add it to your doc

- **Warning:** Do not spend forever on this - it's just a sketch - use circles, squares, and arrows
- **Explanation:** This should just be a sketch of your MVP - what you hope to accomplish by the end of the course - *not* the fully fledged product

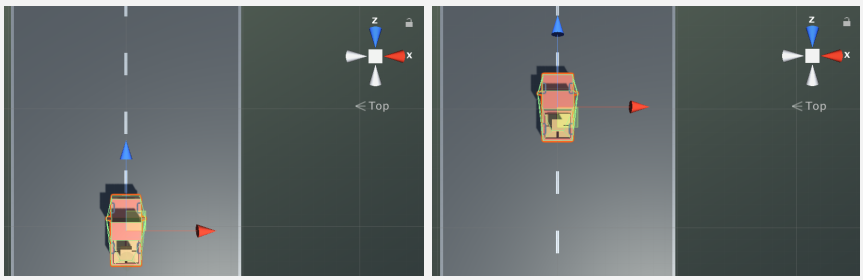


Lesson Recap

New Progress	• Completed your project concept and production timeline
New Concepts and Skills	• Personal Projects • Design Documents • Project Timelines • Project Milestones and Backlogs • Minimum Viable Products

unity

Quiz Unit 1

QUESTION	CHOICES
1 Which Unity window contains a list of all the game objects currently in your scene?	a. Scene view b. Project window c. Hierarchy d. Inspector
2 True or False: Visual Studio is not a part of Unity. You could use a different code editor to edit your C# scripts if you wanted to.	a. True b. False
3 What best describes the difference between the below images, where the car is in the second image is further along the road? 	a. The second car's X location value is higher than the first car's b. The second car's Y location value is higher than the first car's c. The second car's Z location value is higher than the first car's d. The second car's Transform value is higher than the first car's.
4 In what order do you put the words when you are declaring a new variable? `public float speed = 20.0f;`	a. [data type] [access modifier] [variable value] [variable name] b. [access modifier] [data type] [variable name] [variable value] c. [data type] [access modifier] [variable name] [variable value] d. [variable name] [data type]

[access modifier] [variable value]

---

5 Which of the following variables would be visible in the Inspector?

```
public float speed;
float turnSpeed = 45.0f;
private float horizontalInput;
private float forwardInput;
```

a. speed
b. turnSpeed
c. speed & turnSpeed
d. horizontalInput & forwardInput

---

6 What is a possible value for the horizontalInput variable?

```
horizontalInput = Input.GetAxis("Horizontal");
```

a. -10
b. 0.52
c. “Right”
d. Vector3.Up

---

7 What is true about the following two lines of code?

```
transform.Translate(Vector3.forward);
transform.Translate(1, 0, 0);
```

a. They will both move an object the same distance
b. They will both move an object in the same direction
c. They will both move an object along the same axis
d. They will both rotate an object, but along different axes

---

8 Which of the following lines of code is using standard Unity naming conventions?

```
/* a */ Public Float Speed = 40.0f;
/* b */ public float Speed = 40.0f;
/* c */ public float Speed = 40.0f;
/* d */ public float speed = 40.0f;
```

a. Line A
b. Line B
c. Line C
d. Line D

---

9 Which comment would best describe the code below?

```
horizontalInput = Input.GetAxis("Horizontal");
transform.Rotate(Vector3.up, horizontalInput);
```

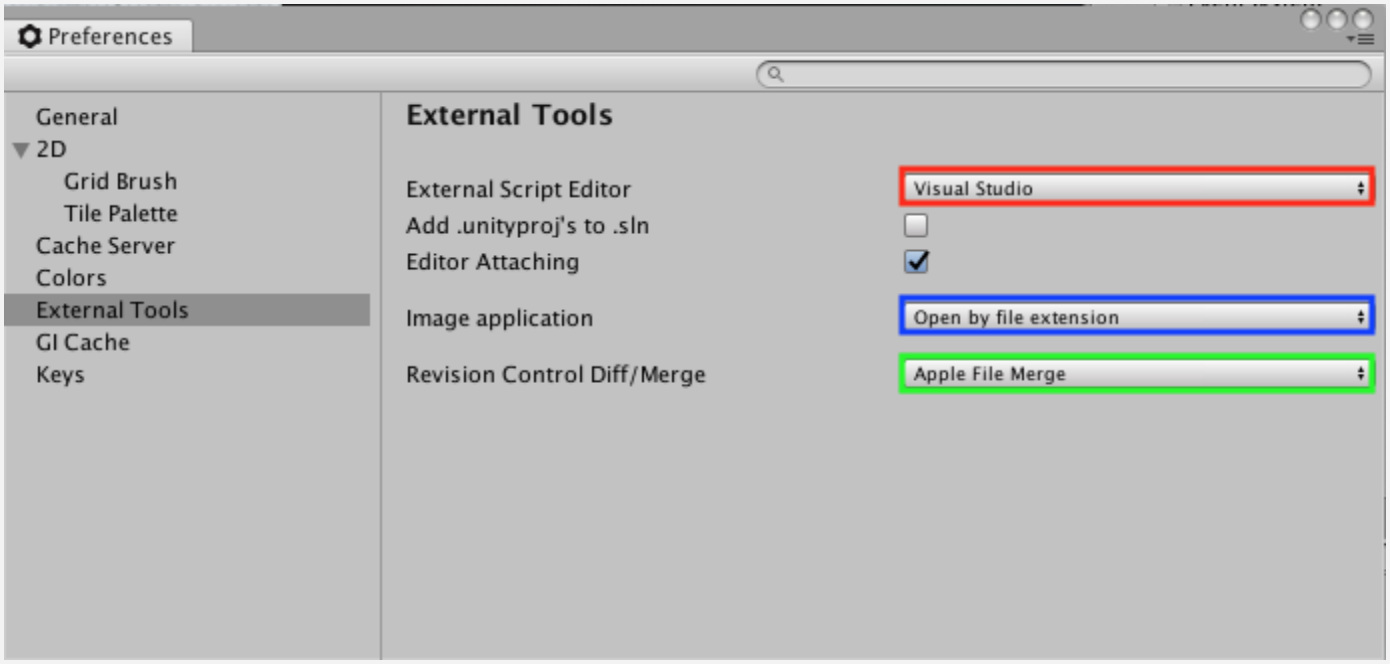
a. // Rotates around the Y axis based on left/right arrow keys
b. // Rotates around the Z axis based on up/down arrow keys
c. // Rotates in an upward direction based on left/right

arrow keys

d. // Moves object up/down based on the the left/right arrow keys

10 The image below shows the preferences window that allows you to change which script editing tool (or IDE) you want to use. Where would you click to choose an alternative code editing tool?

a. The red box (External Script Editor)
b. The blue box (Image application)
c. The green box (Revision control Diff/Merge)



Quiz Answer Key

#	ANSWER	EXPLANATION
1	C	The Hierarchy window contains a list of every GameObject in the current Scene. As objects are added and removed in the Scene, they will appear and disappear from the Hierarchy as well.
2	B	True. Visual Studio is just one of many editors you could use to edit your code, including editors like Atom, Sublime, or even a basic Text Editor.
3	C	You can tell which axis the car has moved along using the XYZ directional gizmo in the top-right, which shows the blue axis pointing forwards down the road.
4	B	Variables are always declared in the order: [access modifier] - public, private, etc [data type] - float, int, GameObject, etc [variable name] - speed, turnSpeed, player, offset, etc [variable value] - 1.0f, 2, new Vector3(0, 1, 0), etc
5	A	“public float speed” would be visible because it has the “public” modifier applied to it
6	B	Input.GetAxis returns a float value between -1 and 1, which means 0.52 is a possible value
7	A	Vector3.forward is the equivalent of (0, 0, 1), which has the same magnitude as (1, 0, 0), even though they’re in different directions, so they would both move an object the same distance, but along different axes
8	D	“public float speed = 40.0f;” uses the correct naming conventions because all three of these terms should start with lowercase letters
9	A	Vector3.up is the Y axis and it’s using the Horizontal input value, so it would rotate around the Y axis when the user presses the left/right arrows
10	A	You would click on the Red box to change the “External Script Editor” from Visual Studio to another tool.

unity

Bonus Features 1 - Share your Work

Steps:

Length: 60 minutes

Overview: In this tutorial, you can go way above and beyond what you learned in this Unit and share what you've made with your fellow creators.

There are four bonus features presented in this tutorial marked as Easy, Medium, Hard, and Expert. You can attempt any number of these, put your own spin on them, and then share your work!

This tutorial is entirely optional, but highly recommended for anyone wishing to take their skills to a new level.

Step 1: Overview

This tutorial outlines four potential bonus features for the Driving Simulation Prototype at varying levels of difficulty:

- **Easy**: Obstacle Pyramids
- **Medium**: Oncoming vehicles
- **Hard**: Camera switcher
- **Expert**: Local multiplayer

Here's what the prototype could look like if you complete all four features:

The Easy and Medium features can probably be completed entirely with skills from this course, but the Hard and Expert features will require some additional research.

Since this is optional, you can attempt none of them, all of them, or any combination in between. You can come up with your own original bonus features as well!

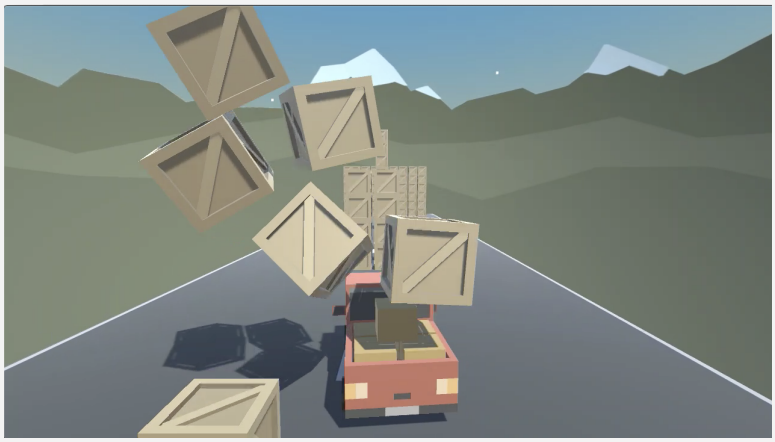
Then, at the end of this tutorial, there is an opportunity to share your work.

We highly recommend that you attempt these using relentless Googling and troubleshooting, but if you do get completely stuck, there are hints and step-by-step solutions available below.

Good luck!

Step 2: Easy: Obstacle pyramids

Create stacks, piles, or pyramids of obstacles for the vehicle to drive through.
These will be more satisfying for the player to crash into compared with single objects.

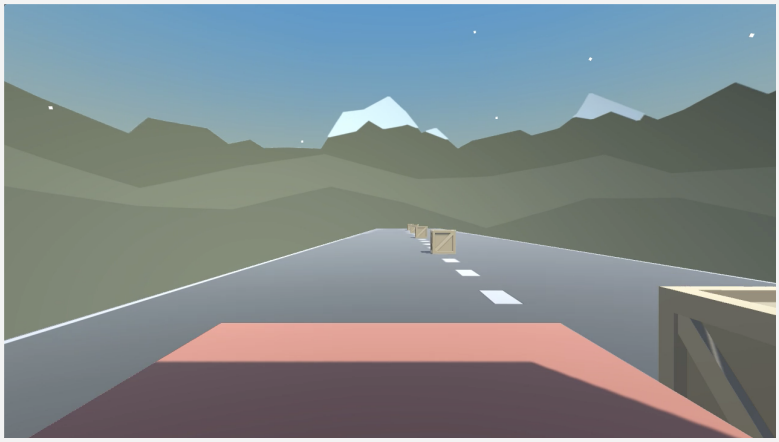


Step 3: Medium: Oncoming vehicles

Add a couple of other cars that are automatically driving down the road in the opposite direction, which the player also has to avoid.
This will make the experience much more challenging, since the player will now be forced to think quickly, rather than taking as much time as they need.

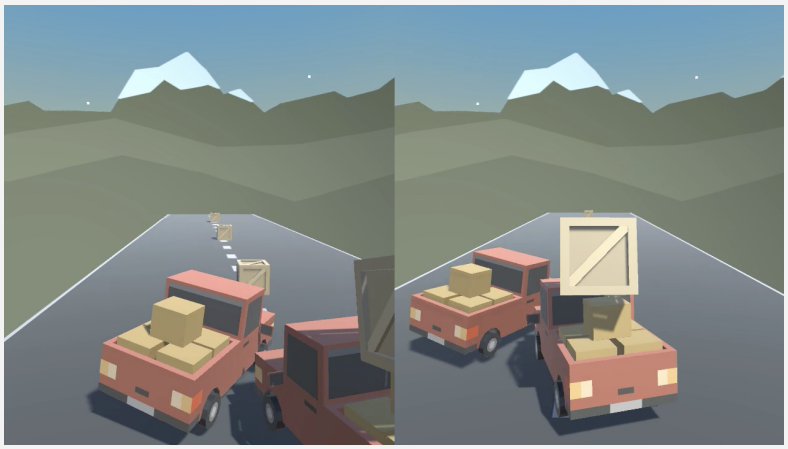
Step 5: Hard: Camera switcher

Allow the player to press a key on the keyboard to switch camera views.
Ideally, the same key would toggle between two views, one above and behind the vehicle, and the other from the perspective of the driver's seat.



Step 6: Expert: Local multiplayer

Transform this into a "local multiplayer" split-screen game with two cars, where one player's car is controlled by WASD and the other is controlled by the arrow keys.
This would add a completely new competitive dimension to the prototype.



Step 7: Hints and solution walkthrough

Hints:

- Easy: Obstacle pyramids
  - Remember to use a Rigidbody!
- Medium: Oncoming vehicles
  - Try using transform.Translate to move the other vehicles.
- Hard: Camera switcher
  - Add a second camera and then use a key press to enable and disable it.
- Expert: Local multiplayer
  - You will need to edit the Input Manager and the Camera's Viewport Rect Width property.

Solution walkthrough
If you are really stuck, download the step-by-step solution walkthrough.
Note that there are likely many ways to implement these features - this is only one suggestion.

Step 8: Share your work

Have you implemented any of these bonus features? Have you added any new, unique features? Have you applied these new features to another project?

We would love to see what you've created!

Please take a screenshot of your project or do a screen-recording walking us through it, then post it here to share what you've made.

We highly recommend that you comment on at least one other creator's submission. What do you like about the project? What would be a cool new feature they might consider adding?